PHP

СРЕДНИЙ УРОВЕНЬ

СОДЕРЖАНИЕ

ПОДКЛЮЧЕНИЕ ВНЕШНИХ ФАЙЛОВ

ВВЕДЕНИЕ

- PHP код необязательно располагать в одном файле - при разработке больших сайтов этот файл может разрастись до невообразимых размеров; кроме того, даже при использовании нескольких больших файлов появляется проблема дублирования кода - например, одна и та же полезная функция может потребоваться сразу в нескольких местах; именно поэтому программы написанные на PHP принято делить на много мелких файлов, которые потом подключатся друг к другу с помощью специальных выражений

- все переменные и функции, которые содержатся в подключаемом файле, будут иметь такую же область видимости как и идентичные им выражения в изначальном файле - никаких особых операторов и модификаторов для обращения к ним не требуется

- Всего существую 4 выражения для подключения внешних файлов: require, require_once, include, include_once

- код, находящийся перед любой из вышеуказанных директив, еще на знает, что находится в подключаемом файле - поэтому обращение к выражениям, которые находятся в этом подключаемом файле, следует размещать после подключения файла

- Если подключаемый внешний файл не будет найден, PHP попытается найти его в специальной директории, которая задается в файле php.ini при помощи настройки include_path

ПРИМЕНЕНИЕ REQUIRE И REQUIRE_ONCE

- выражения require и require_once служат для одной и той же задачи - они пытаются подключить внешний файл; если же этот файл не был найден (как по запрошенному пути, так и в директории, установленной посредством include_path) - произойдет аварийная остановка программы

- Далее рассмотрим стандартный пример с выражением require (предположим, что файлы functions.php и index.php находятся в одной и той же директории):

Файл functions.php

```php
<?php

function calcSumWithTax(float $sum, float $taxPercent): float
{
    return $sum + ($sum * $taxPercent / 100);

}
```

Файл index.php

```php
<?php

require 'functions.php';

echo calcSumWithTax(100, 21); /* функция calcSumWithTax
отсутствует в файле index.php, однако эта функция определена
в файле functions.php, который был подключен несколькими
строчками выше с помощью require 'functions.php'; */
```

- выражение require связано с одной проблемой - если один и тот же файл будет подключен несколько раз, то код находящийся в нем, также будет исполнен несколько раз - это может привести к ошибке, если там, например, объявлена функция (это приведет к аварийной остановке программы):

Файл problem.php

```php
<?php

require 'functions.php';
require 'functions.php'; /* ошибка! - попытка повторного
объявления функции calcSumWithTax */
```

- для решения проблемы require существует схожее выражение require_once - оно делает все то же самое, но оно игнорирует подключение файла, если он уже был ранее подключен:

Файл problem-solution.php

```php
<?php

require_once 'functions.php';
require_once 'functions.php'; // ошибки не будет

echo calcSumWithTax(100, 21); // успешно выведется 121
```

ПРИМЕНЕНИЕ INCLUDE И INCLUDE_ONCE

- выражения include и include_once также пытаются подключить внешний файл; однако если этот файл не был найден - будет показана ошибка, но программа продолжит свое выполнение:

Файл include.php

```
<?php

include 'non-existing-file.php'; /* выведется сообщение об ошибке, но программа не завершится аварийно */

include_once 'non-existing-file.php'; /* еще одно сообщение об ошибке, но программа продолжает работать */

echo 'It\'s alive!'; // строка будет выведена на печать
```

- Во всем остальном выражение include полностью идентично require, а выражение include_once идентично require_once

ВОЗВРАЩЕНИЕ ДАННЫХ ИЗ ПОДКЛЮЧАЕМЫХ ФАЙЛОВ

- каждое выражение, используемое для подключения внешних файлов, возвращает определенное значение - в случае успеха возвращает единица (целое число), а в случае неудачи include и include_once возвращают false (логический тип данных)

- логику возвращения данных из подключаемых файлов можно изменить, если подключаемый файл в глобальной области видимости содержит оператор return с каким-либо значением:

Файл return.php

```php
<?php

$firstNumber = 3;

$secondNumber = 2;

return $firstNumber + $secondNumber; /* возвращение данных из глобальной области видимости */
```

Файл get-return.php

```php
<?php

$returnedValue = require_once 'return.php';

echo $returnedValue; /* на печать будет выведена цифра 5 - т.е. значение полученное из подключенного файла */
```

ЗАГРУЗКА ФАЙЛОВ НА СЕРВЕР: МАССИВ $_FILES

ВВЕДЕНИЕ

- помимо работы с переданными на сервер данными в виде строк и массивов, PHP также умеет обрабатывать загруженные файлы

- один из основных способов загрузки файлов - отправка POST запроса из браузера при помощи html формы

- если при обычном POST запросе в форме можно не указывать способ кодирования передаваемых данных (в этом случае он подставляется автоматически и равняется application/x-www-formurlencoded), то при загрузке файла используется другой способ - multipart/form-data - и его следует указывать явно при помощи атрибута формы enctype:

```
<form
    action="index.php"
    method="POST"
    enctype="multipart/form-data">
        File: <input type="file" name="userfile">
        <br><br>
        <button type="submit">Submit</button>
</form>
```

- дополнительные настройки, которые PHP использует для обработки загружаемых файлов, находятся в файле php.ini:

```
file_uploads = "1" # включает и отключает загрузку файлов
upload_max_filesize = "2M" # максимальный размер загружаемого
файла
post_max_size = "8M" # максимальный размер всего POST запроса
max_file_uploads = 20 # максимальное количество загружаемых
одновременно файлов
upload_tmp_dir = # директория, где файлы будут попадать после
загрузки, но до обработки со стороны PHP; если значения нет,
то берется временная папка операционной системы
max_input_time = "-1" # время, которое отведено PHP для
чтения файла; если -1, то время считается бесконечным
```

ПОЛУЧЕНИЕ ДАННЫХ ЗАГРУЖЕННОГО ФАЙЛА

- после успешной отправки файла на сервер его данные будут вставлены в специальный массив $_FILES; эти данные можно получить по индексу, который идентичен названию того поля формы, куда файл был вставлен (на примере ниже видно, что название поля и индекс массива одно и то же - userfile):

```
<?php

if ($_SERVER['REQUEST_METHOD'] === 'POST' &&
    isset($_FILES['userfile']) &&
    $_FILES['userfile']['error'] === UPLOAD_ERR_OK
) {
    /*
        По ключу userfile массива $_FILES можно получить
        еще один массив, который содержит данные файла
    */
    $file = $_FILES['userfile'];

    echo 'Имя: ', $file['name'], '<br>'; // оригинальное имя

    echo 'Оригинальный путь: ', $file['full_path'], '<br>';
/* путь к файлу на компьютере, с которого загружен файл */

    echo 'Тип: ', $file['type'], '<br>'; // тип файла

    echo 'Код ошибки: ', $file['error'], '<br>'; // код
ошибки (если она произошла)

    echo 'Временный путь: ', $file['tmp_name'], '<br>'; /*
путь к файлу во временной директории */

    echo 'Размер: ', $file['size']; // размер файла (байты)
}
?>

<form
    action="index.php"
    method="POST"
    enctype="multipart/form-data">
        File: <input type="file" name="userfile">
        <br><br>
        <button type="submit">Submit</button>
</form>
```

ОШИБКИ ПРИ ЗАГРУЗКЕ ФАЙЛА

- иногда при загрузке файла происходят ошибки - информацию о них можно получить по индексу error в массиве с данными загружаемого файла

- существует несколько встроенных констант, которые связаны с состоянием файла - сравнивая с ними значения поля по индексу error, можно точно узнать причину ошибки

```
<?php

if ($_SERVER['REQUEST_METHOD'] === 'POST' &&
    isset($_FILES['userfile'])
) {
     // проверка на то, что ошибок нет
     if ($_FILES['userfile']['error'] === UPLOAD_ERR_OK) {
         // здесь обрабатываем успешно загруженный файл
     } else {
         // мы находимся здесь, то ошибка уже точно произошла
         switch ($_FILES['userfile']['error']) {
             case UPLOAD_ERR_CANT_WRITE:
                 echo 'Нельзя записать файл на диск.'; break;
             case UPLOAD_ERR_NO_TMP_DIR:
                 echo 'Нет временной директории.'; break;
             case UPLOAD_ERR_NO_FILE:
                 echo 'Файл не был загружен.'; break;
             case UPLOAD_ERR_PARTIAL:
                 echo 'Файл загружен частично.'; break;
             case UPLOAD_ERR_INI_SIZE:
                 echo 'Превышен допустимый размер.'; break;
         }
     }
}
?>

<form
    action="index.php"
    method="POST"
    enctype="multipart/form-data">
        File: <input type="file" name="userfile">
        <br><br>
        <button type="submit">Submit</button>
</form>
```

СОХРАНЕНИЕ ЗАГРУЖЕННОГО ФАЙЛА

- чтобы не только получить данные файла, но и сохранить его на сервере, следует переместить этот файл из временной директории в какую-либо другую директорию с помощью встроенной функции move_uploaded_file (первый параметр функции - путь ко временной директории, второй параметр - путь, куда будет окончательно перемещен загруженный файл):

```
<?php

if ($_SERVER['REQUEST_METHOD'] === 'POST' &&
    isset($_FILES['userfile']) &&
    $_FILES['userfile']['error'] === UPLOAD_ERR_OK
) {
    // загрузка файла
    $success = move_uploaded_file(
        $_FILES['userfile']['tmp_name'],
        './upload/userfile.txt'
    );

    if ($success) {
        echo 'Файл был успешно сохранен.';
    } else {
        echo 'При сохранении файла произошла ошибка.';
    }
}
?>

<form
    action="index.php"
    method="POST"
    enctype="multipart/form-data">
        File: <input type="file" name="userfile">
        <br><br>
        <button type="submit">Submit</button>
</form>
```

ТИПЫ ДАННЫХ 6: РЕСУРС (RESOURCE)

ВВЕДЕНИЕ

- в PHP существует особый тип данных, который не может быть обработан, получен в 'сыром' виде (например, как единица 1 или строка 'привет') или выведен на печать - это ресурс

- ресурс представляет собой фактически ссылку на некий внешний источник - например, поток данных из файла или соединение с базой данных

- Ресурс не может быть обработан стандартными средствами - для каждого вида ресурса существует свой набор функций; в этом уроке в качестве примера выбрано взаимодействие с потоком данных из файла

ВЗАИМОДЕЙСТВИЕ С ПОТОКОМ ДАННЫХ ИЗ ФАЙЛА

- По своей сути взаимодействие с потоком данных из файла является примером низкоуровневой работы с файлом (PHP предлагает ряд удобных функций, но внутри все они содержат логику, о которой рассказывается ниже):

```
<?php

/*
    Файловый ресурс можно получит в разных контекстах - эти
    Контексты определяются вторым параметром функции fopen:
    r  получение ресурса только для чтения
    r+ для чтения и записи
    a  для записи (курсор помещается в конец файла)
    a+ для чтения и записи (курсор помещается в конец файла)
    w  для записи (перед получением ресурса файл очищается)
    w+ для чтения и записи (перед получением ресурса файл
очищается)
*/

/*
    откроем файл для чтения - если открыть не удастся -
    fopen вернет не ресурс а логический тип false
*/
$res = fopen('filename.txt', 'r');

// выведем построчно все строки файла
while (!feof($res)) { // пока не достигнут конец файла …
    // каждое чтение смещает курсор
    echo fgets($res);
}

rewind($res); // устанавливаем курсор в начало потока

// ресурс следует закрывать после работы с ним
fclose($res);

// ---------------------------------------------------------->

// откроем файл для записи
$src = fopen('file-for-writing.txt', 'w');

fwrite($src, 'Привет!'); // запись в файловый поток

fclose($src); // закроем ресурс
```

ТИПЫ ДАННЫХ 7: NULL

- иногда при работе программы возникают неопределенные ситуации, которые не так легко решить стандартными средствами, например, пропала связь с базой данных - в таких случаях не всегда надо аварийно завершать программу, иногда стоит попробовать еще раз, поискать альтернативы и т.д.

- чтобы обозначить такое отсутствие результата и записать в переменную нечто, имеющее смысл, существует тип данных null, который имеет одно единственное значение - null

```php
<?php

$result = null;
```

- функции, которые имеют в качестве возвращаемого типа void, на самом деле возвращают null

```php
<?php

// на печать будет выведено NULL
echo gettype(example());

function example(): void
{
}
```

- при объявлении функции существует возможность указать такой “двойственный” тип параметров или “двойственный” тип возвращаемого значения (например, строка или null, либо целое число или null и т.п.) - это делается при помощи знака ? перед объявлением типа:

```php
<?php

// каждый параметр может принять число или null
function concatIntegers(?int $a, ?int $b): ?string
{
    // должна вернуться строка или null
    return ($a == null || $b == null) ? null : $a . $b;
}
```

- в математических операциях null ведет себя как 0:

```
<?php

echo 10 + null; // будет выведено 10
```

- в операциях со строками null автоматически преобразуется в пустую строку:

```
<?php

echo 'Hello' . null; // будет выведено Hello
```

ФУНКЦИИ ДЛЯ РАБОТЫ С ФАЙЛАМИ И ДИРЕКТОРИЯМИ

ОСНОВНЫЕ ФУНКЦИИ ДЛЯ РАБОТЫ С ФАЙЛАМИ

```php
<?php

// проверка на существование файла или директории
if (file_exists('somename')) {
    echo 'Файл или директория существует.';
}

// проверка на существование файла
if (is_file('filename.txt')) {
    echo 'Файл существует.';
}

// получение содержимого файла в виде строки
file_get_contents('filename.txt');

// запись в файл (файл будет полность перезаписан)
file_put_contents('filename.txt', 'some text');

// запись в файл (текст будет добавлен в конец файла)
file_put_contents('filename.txt', 'some text', FILE_APPEND);

// получение размера файла (в байтах)
filesize('filename.txt');

// переименование файла
rename('old-name.txt', 'new-name.txt');

// копирование файла
copy('old/file.txt', 'new/file.txt');

// получение строк файла в виде массива
file('filename.txt');

// удаление файла
unlink('filename.txt');
```

ОСНОВНЫЕ ФУНКЦИИ ДЛЯ РАБОТЫ С ДИРЕКТОРИЯМИ

```php
<?php

// проверка на существование директории
if (is_dir('some-directory-name')) {
    echo 'Директория существует.';
}

// создание директории
mkdir('directory-name');

// удаление директории (только если директория пустая)
rmdir('directory-name');

/*
    получение файлов и директорий внутри запрошенной
    директории в виде массива - в числе прочего содержит
    запрошенную директорию (в виде точки - .), а также
    родительскую директорию (в виде двоеточия - ..)
*/
scandir('directory-name');

// напечатает все вложенные файлы и директории
printDirContent('directory-name');

// функция для чтения всех вложенных файлов и директорий
function printDirContent(string $dirname): void
{
    foreach (scandir($dirname) as $item) {
        if ($item === '.' || $item === '..') {
            continue;
        }

        // печатаем найденную директорию или файл
        $path = $dirname . '/' . $item;
        echo $path, '<br>';

        // если директория - обходим и ее
        if (is_dir($path)) {
            printDirContent($path);
        }
    }
}
```

РАБОТА С COOKIES

ВВЕДЕНИЕ

- при работе сайта существует необходимость отслеживать деятельность посетителей - проверять был залогинен человек или нет, посещал он ту или иную страницу, когда он последний раз был на сайте и т.д.

- для решения этой проблемы существует механизм cookies - данных, которые сайт может попытаться сохранить на компьютере пользователя (но компьютер не обязан их сохранять)

- cookies устанавливаются вместе с HTTP ответом при помощи заголовка Set-Cookie; затем при каждом заходе на сайт в HTTP запрос браузер будет вставлять заголовок Cookie, который и будет содержать сохраненные для этого сайта данные

- cookies имеют срок годности - если при установке не был установлен срок хранения, то cookies на компьютере будут удалены после закрытия браузера, если же срок хранения был установлен, то cookies будут удалены только тогда, когда этот срок истечет

- кроме срока годности, cookies также имеет ряд других необязательных модификаторов: Path - путь на сайте, для которого будут доступны cookies, secure - cookies будут передаваться только по протоколу HTTPS, httponly - JavaScript не будет иметь доступа к cookies, domain - позволяет определить будут ли доступны cookies на домене и всех поддоменах или только на конкретном поддомене

ПРИМЕРЫ HTTP ЗАПРОСОВ С УСТАНОВКОЙ И ПОЛУЧЕНИЕМ COOKIES

- пример HTTP ответа сервера, при помощи которого сайт просит браузер установить в качестве cookies два значения - firstname и lastname (в обоих случаях cookies устанавливаются на 1 час - применен модификатор Max-Age):

```
HTTP/1.1 200 OK
Content-Type: text/html
Content-Length: 12345
Set-Cookie: firstname=John; Max-Age=3600
Set-Cookie: lastname=Smith; Max-Age=3600

<!DOCTYPE html>
<html>
...
</html>
```

- Пример HTTP запроса на сервер, в котором видны ранее установленные cookies:

```
GET /index.php HTTP/1.1
Host: www.example.com
Accept: text/html
Accept-Language: ru
Cookie: firstname=John; lastname=Smith

User-Agent: Mozilla/5.0 ...
```

УПРАВЛЕНИЕ COOKIES ПРИ ПОМОЩИ PHP

- для установки cookies в PHP существует одна единственная функция setcookie - она имеет много параметров, но обязательным является только первый - name (если применить этот минимум, то будет установлена одна cookie, с переданным именем и пустым значением - эта cookie будет удалена после закрытия браузера):

```
<?php

setcookie('someName');
```

- однако смысла в таких данных немного, поэтому обычно для cookie также устанавливают значение (в данном случае John) и срок годности (1 день - 86400 секунд):

```
<?php

setcookie('firstname', 'John', time() + 86400);
```

- Но можно установить и полный набор параметров функции setcookie:

```
<?php

$name = 'firstname';
$value = 'John';
$expires = time() + 86400;
$path = '/';
$domain = 'example.com';
$secure = true;
$httponly = true;

setcookie(
    $name,
    $value,
    $expires,
    $path,
    $domain,
    $secure,
    $httponly
);
```

- PHP предоставляет возможность не только устанавливать cookies, но и читать их - все доступные cookies находятся в глобальном ассоциативном массиве $_COOKIE (имена cookies в нем представлены в качестве строковых ключей) - если мы хотим, проверить, что при предыдущем ответе была установлена cookie firstname, то правильно выполнить это можно следующим образом:

```
<?php

if (isset($_COOKIE['firstname']) &&
    !is_array($_COOKIE['firstname']) // проверка на не-массив
) {
    // для безопасности применим strip_tags
    echo strip_tags($_COOKIE['firstname']);
}
```

- Для принудительного удаления cookie необходимо отправить данные со сроком годности, который установлен в прошлом:

```
<?php

setcookie('firstname', 'John', time() - 1);
```

РАБОТА С СЕССИЯМИ

ВВЕДЕНИЕ

- cookies - неплохой инструмент для контроля состояния посетителей сайта, но он имеет два существенных недостатка - во-первых, для каждого конкретного сайта размер данных в cookies не должен превышать 4 килобайт, во-вторых, данные хранятся на компьютере пользователя - он может их читать и менять по своему усмотрению - если в cookies хранить важные данные, то может пострадать как безопасность посетителя, так и безопастность сайта

- для решения проблем с cookies в PHP применяется механизм сессий - у посетителя хранится только одна cookie с уникальным для этого посетителя идентификатором; на сервере для этого значения хранятся данные (обычно в файле, название которого идентично идентификатору сессионной cookie) - т.к. идентификатор уникален, для каждого посетителя можно хранить свои уникальные данные

- посетитель может вручную изменить значение сессионной cookie, но тогда данные, хранящиеся на сервере, больше не будут ассоциироваться с ним

- по умолчанию сессионная cookie имеет название PHPSESSID (т.е. посетителю будет установлена cookie с именем PHPSESSID и значением - уникальным идентификатором), но это название можно поменять при помощи файла php.ini и его настройки session.name

- срок жизни сессии по умолчанию кончается тогда, когда пользователь закрывает браузер, но это поведение можно изменить - файле php.ini есть настройка session.cookie_lifetime - с ее помощью можно установить то количество секунд, которое и будет жить сессия

УПРАВЛЕНИЕ СЕССИЯМИ ПРИ ПОМОЩИ PHP

- чтобы создать сессию или возобновить уже существующую используется функция session_start()

- после возобновления или создания сессии становится доступным глобальный ассоциативный массив $_SESSION - в отличие от cookies, вся работа с данными (вставка новых, обновление и удаление старых) происходит только при взаимодействии с этим массивом:

```
<?php

session_start();

// установка данных сессии
$_SESSION['firstname'] = 'John';
$_SESSION['lastname'] = 'Smith';
```

- при повторном заходе на тот же сайт ранее установленные данные сессии становятся доступными вновь:

```
<?php

session_start();

if (isset($_SESSION['firstname']) &&
    isset($_SESSION['lastname'])
) {
    // на печать будет выведено John Smith
    echo $_SESSION['firstname'].' '.$_SESSION['lastname'];
}
```

- иногда появляется необходимость необратимо уничтожить все данные сессии - для этого необходимо использовать функцию session_destroy():

```
<?php

session_start();

// уничтожение сессии
session_destroy();
```

ТИПЫ ДАННЫХ 8: ОБЪЕКТЫ

ВВЕДЕНИЕ

- один из важных и одновременно самых сложных типов данных в PHP - объект, использование которого является попыткой мыслить о программировании в категориях реального мира

- использование объектов не делает код быстрее, но облегчает разработку больших приложений и ускоряет внедрение изменений в вашу программу, т.е. позволяет программисту лучше контролировать свой код

- сам по себе объект является совокупностью свойств (переменных) и методов (функций) для обработки этих свойств

- объект всегда создается при помощи шаблона, который называется классом - это гарантирует то, что все объекты такого класса-шаблона будут иметь одинаковую структуру (одинаковые методы и одинаковые типы свойств, но значения этих свойств будут разными)

- PHP позволяет контролировать доступ к методам и свойствам объекта - некоторые из этих вещей можно сделать доступными только для методов этого же объекта, если их использование извне влечет за собой нарушение какой-либо внутренней логики

СОЗДАНИЕ ОБЪЕКТОВ

- как уже было сказано, для создания объекта нужен класс - некий шаблон, который задает структуру будущего объекта:

```php
<?php

class User
{
    /*
        создание строкового атрибута (модификатор public
        говорит о том, что атрибут будет доступен снаружи
        объекта, который будет создан с помощью этого класса)
    */
    public string $firstname;
    public string $lastname;
    public string $password;

    // создание метода
    public function getFullName(): string
    {
        /*
            использование ключевого слова $this означает,
            что мы обращаемся к атрибуту, значение
            которого будет присвоено после создания объекта
        */
        return $this->firstname . ' ' . $this->lastname;
    }
}
```

- создание объекта при помощи класса осуществляется при помощи ключевого слова new - после этого объекту можно назначить значения атрибутов и обращаться к ним:

```php
<?php

// предположим, что нам доступен класс User

$user = new User(); // создание объекта из класса
$user->firstname = 'John'; // назначение значения атрибута
$user->lastname = 'Smith';
$user->password = 'secret';

echo $user->password; // обращение к атрибуту - будет
напечатано secret
echo $user->getFullName(); // вызов метода - будет напечатано
John Smith
```

- предыдущий пример не навязывает присвоение значений атрибутов - это создает потенциальную угрозу того, что может произойти обращение к неинициализированному атрибуту - в результате программа будет аварийно остановлена; чтобы это предотвратить, надо применять т.н. констуктор - метод, который автоматически вызывается при создании объекта:

```php
<?php

class User
{
    public string $firstname;
    public string $lastname;
    public string $password;

    /*
        конструктор автоматически вызывается при создании
        объекта и потребует, чтобы в него передали три
        параметра
    */
    public function __construct(
        string $firstname,
        string $lastname,
        string $password
    ) {
        $this->firstname = $firstname;
        $this->lastname = $lastname;
        $this->password = $password;
    }

    public function getFullName(): string
    {
        return $this->firstname . ' ' . $this->lastname;
    }
}

// теперь объект следует создавать с передачей параметров
$user = new User('John', 'Smith', 'secret');
echo $user->password; // будет напечатано secret
echo $user->getFullName(); // будет напечатано John Smith

// создание еще одного значение с другими значениями
$user2 = new User('Ivan', 'Ivanov', 'password');
echo $user2->password; // будет напечатано password
echo $user2->getFullName(); // будет напечатано Ivan Ivanov
```

НАСЛЕДОВАНИЕ

- в PHP доступно наследование классов - класс наследник получает все методы и типы атрибутов класса родителя (но может добавить к ним и свои методы и атрибуты) - можно наследоваться только от одного класса

- осуществляется при помощи ключевого слова extends:

```php
<?php

// предположим, что нам доступен класс User

// наследование при помощи extends
class Admin extends User
{
    // переопределение родительского метода
    public function getFullName(): string
    {
        /*
            обращение к методу класса родителя при помощи
            ключевого слова parent
        */
        return 'администратор ' . parent::getFullName();
    }
}

class Client extends User
{
    // добавление нового атрибута
    public array $orderedProducts = [];

    // переопределение конструктора
    public function __construct(
        string $firstname,
        string $lastname,
        string $password,
        array  $orderedProducts
    ) {
        parent::__construct(
            $firstname,
            $lastname,
            $password
        );
        $this->orderedProducts = $orderedProducts;
    }
}
```

- наследование позволяет нам гарантировать, что у всех классов наследников будут методы и типы атрибутов класса родителя:

```php
<?php

// предположим, что нам доступны классы User, Admin, Client

// создадим массив, где будут администраторы и клиенты
$visitors = [
    new Admin('John', 'Doe', '12345678'),
    new Client('Mary', 'Sue', 'secret', []),
];

/*
    функция принимает только объекты класса User и
    наследников класса User - мы знаем, что User имеет метод
    getFullName, поэтому мы может спокойно его вызывать
*/
function getUserName(User $user): string
{
    return $user->getFullName();
}

$message = '';
foreach ($visitors as $visitor) {
    if ($message === '') {
        $message .= getUserName($visitor);
    } else {
        $message .= ', ' . getUserName($visitor);
    }
}

/*
    Будет выведено: Сейчас на сайте - администратор John Doe,
    Mary Sue
*/
echo 'Сейчас на сайте - ' . $message;
```

- говоря о наследовании, стоит упомянуть модификаторы доступа - ранее уже говорилось, что модификатор public предоставляет полный доступ к атрибутам и методам как внутри, так и снаружи объекта, но есть еще два модификатора - private (такие методы и атрибуты доступны только в том классе, где они объявлены, но не снаружи объекта и не в классах наследниках) и protected (доступ возможен в классе, где метод или атрибут объявлен, а также в классах наследниках)

НАСЛЕДОВАНИЕ ОТ АБСТРАКТНЫХ КЛАССОВ

- в PHP существуют так называемые абстрактные классы - на их основе нельзя создать объект, т.к. они обычно содержат нереализованные прототипы методов - т.н. абстрактные методы (основное правило - если в классе есть абстрактные методы, то класс должен быть объявлен абстрактным)

- чтобы использовать абстрактный класс, на сделать для него класс-наследник, переопределить все абстрактные методы, а уже затем создать объекты из классов наследников

- основная задача - уменьшить дублирование кода

```php
<?php

abstract class Page // объявление абстрактного класса
{
    public function getHeader(): string
    {
        return 'I am site header!<br>';
    }

    // объявление абстрактного метода - у него нет тела!
    abstract public function getContent(): string;

    public function getFooter(): string
    {
        return 'I am site footer!';
    }
}

class Main extends Page
{
    public function getContent(): string
    {
        return 'I am site main page content!';
    }
}

class Blog extends Page
{
    public function getContent(): string
    {
        return 'Hello, I am site blog page with articles!';
    }
}
```

- применять абстрактные классы можно следующим образом:

```php
<?php

// предположим, что нам доступны классы Page, Main, Blog

/*
    объявление функции, которая рисует страницы сайта -
    она примет любой объект, который создан при помощи
    классов наследников класса Page
*/
function drawSitePage(Page $page): string
{
    return
        $page->getHeader() .
        $page->getContent() .
        $page->getFooter();
}

// выбираем класс в зависимости от запрошенной ссылки
if ($_SERVER['SCRIPT_NAME'] === '/blog') {
    $page = new Blog();
} else {
    $page = new Main();
}

// рисуем страницу сайта
echo drawSitePage($page);
```

ИСПОЛЬЗОВАНИЕ ИНТЕРФЕЙСОВ

- в PHP есть ещё более абстрактные вещи, чем абстрактные классы - это интерфейсы, у которых все методы обязаны не иметь тела, а атрибутов не должно существовать в принципе

- интерфейсы служат для того, чтобы классы, которые имплементированы при помощи таких интерфейсов, имели один и тот же набор методов и это было бы гарантированно в любой точке программы

- объявление интерфейсов похоже на объявление классов, только вместо слова class используется слово interface

- классы, которые имплементируются от интерфейсов (при помощи ключевого слова implements) обязаны переопределить все методы интерфейса

```php
<?php

// объявление интерфейса
interface Report
{
    // методы интерфейса не могут иметь тела
    public function generateReport(): string;
}

// имплементация класса от интерфейса
class XMLReport implements Report
{
    // переопределение метода интерфейса
    public function generateReport(): string
    {
        return '<root><id>725</id><price>150</price></root>';
    }
}

// имплементация класса от интерфейса
class JSONReport implements Report
{
    // переопределение метода интерфейса
    public function generateReport(): string
    {
        return '{"id": 725, "price": 150}';
    }
}
```

ИСПОЛЬЗОВАНИЕ ТРЕЙТОВ

- иногда складывается ситуация, когда класс уже является наследником какого-либо класса, но одновременно ему необходима логика, которая используется в другом классе - наследоваться опять уже невозможно, т.к. в PHP не разрешено множественное наследование

- PHP позволяет решить эту проблему при помощи трейтов - совокупности методов и атрибутов, которая может быть подключена к разным классам без наследования

- трейт объявляется при помощи ключевого слова trait, а подключается к классу при помощи ключевого слова use:

```php
<?php

// объявление трейта
trait TaxCalculation
{
    private float $taxPercent = 21;

    public function calculateSumWithTax(float $price): float
    {
        return $price + ($price / 100 * $this->taxPercent);
    }
}

class Product
{
    // подключение трейта - если их будет несколько,
    // то можно перечислить все подряд через запятую
    use TaxCalculation;

    public float $basePrice = 99.99;

    public float $quantity = 10;

    public function getTotalToPay(): float
    {
        $singleProductSumWithTax = $this->
            calculateSumWithTax($this->basePrice);

        return $this->quantity * $singleProductSumWithTax;
    }
}
```

СТАТИЧЕСКИЕ МЕТОДЫ И АТРИБУТЫ КЛАССА

- до этого момента основная работа была связана с объектами, а классы использовались только в качестве шаблонов - однако в PHP и классы могут иметь свои собственные атрибуты, методы, а также то, чего нет у классов - константы

- методы объекта могут использовать методы и атрибуты класса, но методы класса не могут использовать методы и атрибуты объекта

- все вышеупомянутые элементы становятся привязанными к классу (а не к объекту этого класса) посредством использования ключевого слова static

```php
<?php

class Math
{
    // объявление константы
    public const PI = 3.14;

    public static int $number = 123;

    public static function getCircleArea(
        float $radius
    ): float {
        /*
            обращение к статическим методам и атрибутам
            класса производится при помощи слова self
        */
        return self::PI * ($radius ** 2);
    }

    public static function getNumber(): int
    {
        return self::$number;
    }
}

echo Math::PI, '<br>'; // обращение к классу - выведется 3.14

echo Math::getCircleArea(10), '<br>'; // выведется 314

echo Math::$number; // 123
```

РАБОТА С БАЗОЙ ДАННЫХ

НАСТРОЙКА БАЗЫ ДАННЫХ (СУБД MYSQL)

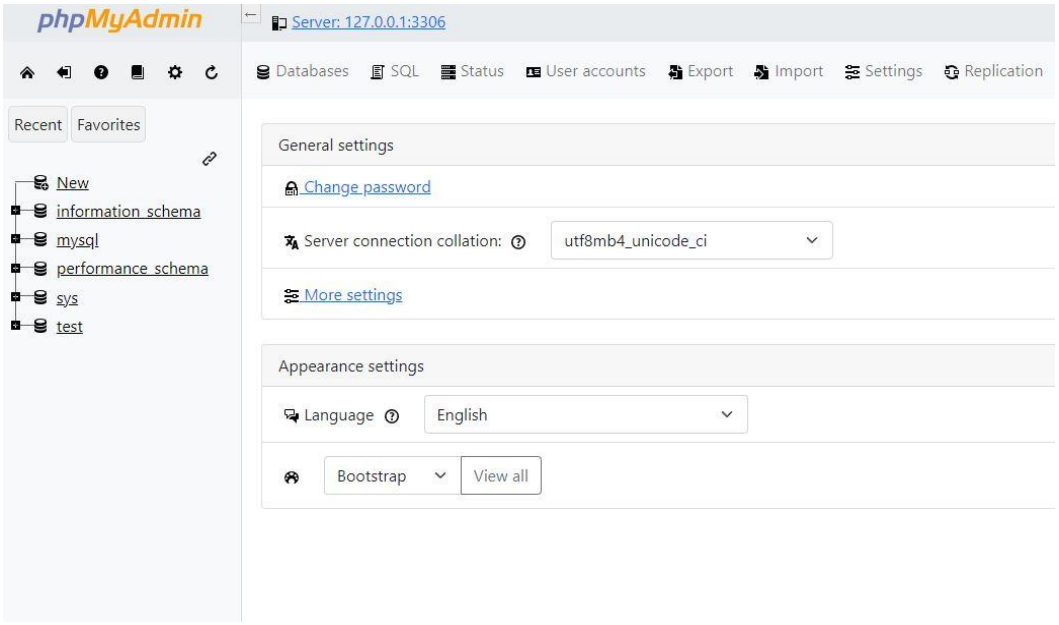
- для работы с базами данных сначала необходимо включить сервер с работающей на нем системой управления этими базами данных (СУБД) - в нашем случае это будет MySQL

- сначала убедимся, что у нас включен OpenServer (если не работает, то включим его - должен загореться зеленый флажок)

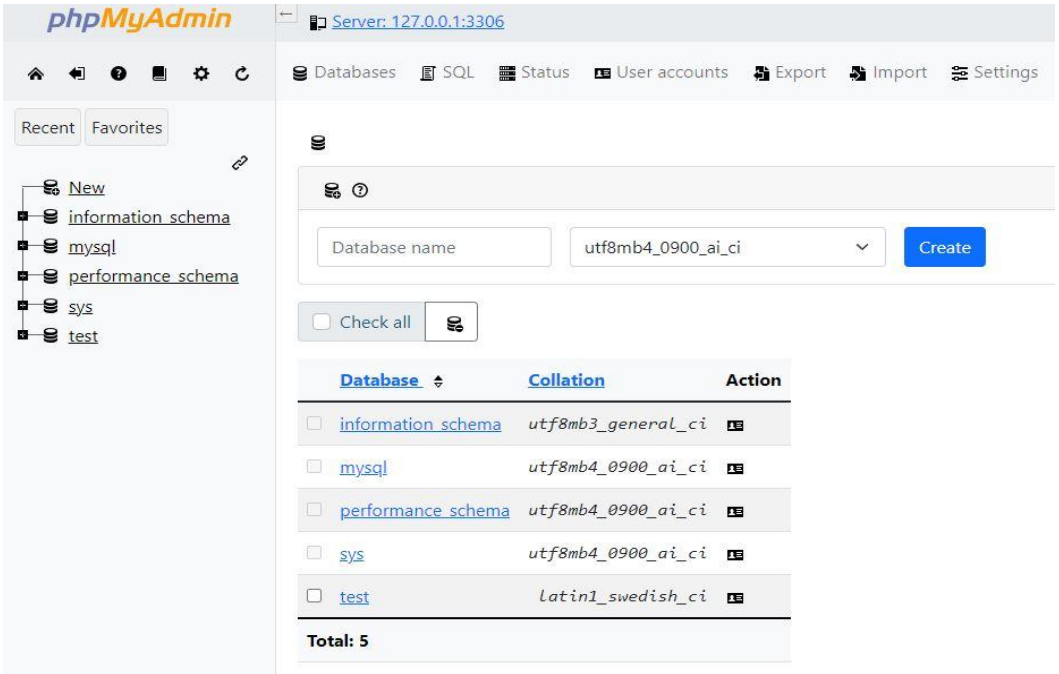
- переходим в браузере по ссылке: http://127.0.0.1/openserver/phpmyadmin/index.php и видим следующую страницу:

- для продолжения в поле Username пишем **root**, а поле Password оставляем пустым - после этого нажимаем на кнопку **Log in**

- на новой странице нажимаем на кнопку **New** в верхнем левом углу (это необходимо для создания нашей базы данных):



- В открывшемся окне в поле Database name пишем **php**, а рядом выбираем кодировку **utf8mb4_0900_ai_ci** - затем нажимаем на кнопку **Create** - если все прошло успешно, то поздравляю - вы создали свою первую базу данных!



УПРАВЛЕНИЕ ДАННЫМИ ПРИ ПОМОЩИ ЯЗЫКА SQL (СУБД MYSQL)

- язык SQL позволяет управлять теми значениями, которые хранятся в базе данных - его область применения можно разделить на две части - во-первых, это создание самой структуры базы данных, таблиц, колонок и их типов (Data Definition Language), во-вторых, это манипуляция данными (получение, вставка, редактирование, удаление) (Data Modification Language)

- наиболее часто используемые типы данных в MySQL:

Тип	Описание	От	До
INT	Целое число	-2147483648	2147483647
BIGINT	Большое целое число	-9.22337E+18	9.22337E+18
BOOLEAN	Логический тип	0	1
FLOAT	Дробное число	1.175E-38	3.4E+38
DOUBLE	Большое дробное число	2.23E-308	1.8E+308
VARCHAR	Строка		Обычно до 255 символов, но может доходить до 65535 байт
TEXT	Длинный текст		до 65535 байт
LONGTEXT	Очень длинный текст		До 4 гигабайт
DATETIME	Дата и время	1001-01-01 00:00:00	9999-12-31 23:59:59
DATE	Дата	1001-01-01	9999-12-31
TIME	Время	-838:59:59	838:59:59
TIMESTAMP	Временная метка (кол-во секунд от начала 1970 года)	1970-01-01 00:00:00	2037 год

- помимо типов данных в базах данных есть такое понятие как ключи - из них основными являются первичный ключ (PRIMARY KEY - колонка, которая является уникальным идентификатором ряда таблицы - в таблице не может быть двух ключей с одинаковым значением), а также внешний ключ (FOREIGN KEY - колонка, которая содержит ссылку на некое значение в другой таблице)

- первичный ключ чаще всего не надо контролировать вручную - при создании таблицы следует добавить к числовому полю атрибут AUTO_INCREMENT (если мы хотим, чтобы именно это поле было первичным ключом) - тогда ключ будет автоматически добавляться и обновляться

- таблицы создаются следующим образом (ENGINE - движок таблицы (необязательное значение), DEFAULT CHARSET - кодировка текстовых полей по умолчанию):

```
CREATE TABLE IF NOT EXISTS products (
    id INT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(255) NOT NULL,
    description TEXT,
    price FLOAT NOT NULL
)ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

- далее пример создания таблицы с внешним ключом product_id (который будет ссылаться первичный ключ уже созданной таблицы products - колонку id):

```
CREATE TABLE IF NOT EXISTS product_attributes (
    name VARCHAR(255) NOT NULL,
    value VARCHAR(255) NOT NULL,
    product_id INT UNSIGNED NOT NULL,
    FOREIGN KEY (product_id) REFERENCES products(id) ON
DELETE CASCADE
)ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

- для вставки данных в таблицу используется оператор INSERT (если это первая вставка в таблицу, то id автоматически получит значение 1, вторая вставка - 2 и т.д.) - строковые данные следует помещать в кавычки:

```
INSERT INTO products(
    name, description, price
) VALUES (
    'Smartphone iPhone', 'Very nice smartphone!', 999.99
);
```

```
INSERT INTO products(
    name, description, price
) VALUES (
    'Smartphone Samsung', 'Another nice smartphone!', 859.99
);
```

- можно вставить несколько рядов одновременно (перечислив их через запятую):

```
INSERT INTO product_attributes (
    name, value, product_id
) VALUES ('color', 'gray', 1), ('camera', '48 MP', 1);
```

- чтобы получить данные из таблицы используется оператор SELECT (чтобы получить конкретные колонки - перечисляем их через запятую, а чтобы получить все колонки без перечисления - используем звездочку - *):

```
SELECT id, name, description, price FROM products;
```

- чтобы получить не все ряды, а только те ряды, которые соответствуют определенным критериям, используется оператор WHERE (если условий несколько, то они перечисляются с помощью ключевого слова AND):

```
SELECT
    id, name, description, price
FROM products
WHERE id = 1
AND name = 'Smartphone iPhone';
```

- Иногда в одном запросе требуется связать данные двух таблиц одновременно - для этого используется оператор JOIN (если применяется просто JOIN - данные вернутся, если условие будет выполняться для обеих таблиц одновременно, LEFT JOIN - если хотя бы для первой таблицы - вернутся данные из первой таблицы, а все недостающие данные из второй таблицы вернутся со значением NULL, RIGHT JOIN - если условие выполняется по крайней мере для второй таблицы):

```
SELECT
    p.name,
    p.description,
    pa.name AS product_attribute_name,
    pa.value AS product_attribute_value
FROM products p
LEFT JOIN product_attributes pa
    ON p.id = pa.product_id;
```

- Для обновления значений колонок используется оператор UPDATE (в данном примере будут обновлены те ряды, у которых id = 1):

```
UPDATE
    products
SET
    name = 'Smartphone iPhone (Discounted)',
    price = price - 100
WHERE id = 1;
```

- для удаления рядов, соответствующих каким-либо критериям, применяется оператор DELETE в связке с оператором WHERE:

```
DELETE FROM products WHERE id = 1;
```

- если необходимо удалить все ряды из таблицы - применяется оператор DELETE без каких-либо дополнений:

```
DELETE FROM products;
```

ВЗАИМОДЕЙСТВИЕ PHP И БАЗ ДАННЫХ (СУБД MYSQL)

- в PHP существуют два основных способа работы с базой данных - использование модуля mysqli и использование модуля PDO

- Наиболее распространенным на данный момент является модуль PDO, т.к. он умеет работать не только с СУБД MySQL, но и другими СУБД (PostgreSQL, SQLite и т.д.) - чтобы убедиться, что он подключен, необходимо в конфигурационном файле php.ini проверить расширение pdo_mysql (extension=pdo_mysql) - оно не должно быть закомментировано

- Для того, чтобы начать работать с базой данных в PHP, сначала необходимо установить соединение - если предположить, что у нас уже создана база данных из предыдущего примера (но для нормального продолжения обучения следует удалить из нее все таблицы), то код выглядит следующим образом:

```php
<?php

$url = 'mysql:host=localhost;dbname=php;charset=utf8';
$username = 'root';
$password = '';

$connection = new PDO($url, $username, $password, [
    // устанавливаем, что при ошибке будет вызвано исключение
    PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION,

    // при выборках мы будем получать ассоциативный массив
    PDO::ATTR_DEFAULT_FETCH_MODE => PDO::FETCH_ASSOC
]);
```

- Создание таблиц при помощи PDO:

```php
<?php

// предположим, что нам уже доступна переменная $connection

// запрос для создания таблицы products
$query = 'CREATE TABLE IF NOT EXISTS products (
    id INT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(255) NOT NULL,
    description TEXT,
    price FLOAT NOT NULL
)ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;';

// подготавливаем соединение к запросу
$productsCreateStatement = $connection->prepare($query);

// исполняем запрос - таблица products создана!
$productsCreateStatement->execute();

// ---------------------------------------------------------->

// запрос для создания таблицы product_attributes
$query = 'CREATE TABLE IF NOT EXISTS product_attributes (
    name VARCHAR(255) NOT NULL,
    value VARCHAR(255) NOT NULL,
    product_id INT UNSIGNED NOT NULL,
    FOREIGN KEY (product_id) REFERENCES products(id) ON
DELETE CASCADE
)ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;';

// подготавливаем соединение к запросу
$attrsCreateStatement = $connection->prepare($query);

// исполняем запрос - таблицы product_attributes создана!
$attrsCreateStatement->execute();
```

- вставка данных:

```
<?php

// предположим, что нам уже доступна переменная $connection

// запрос для вставки данных
$query = 'INSERT INTO products(
    name, description, price
) VALUES (
    :name, :description, :price
);';

// подготавливаем соединение к запросу
$insertStatement = $connection->prepare($query);

// исполняем запрос
$insertStatement->execute([
    'name'        => 'Smartphone iPhone',
    'description' => 'Very nice smartphone!',
    'price'       => 999.99
]);

/*
    очень часто нам необходимо получить значение
    первичного ключа вставленной записи - для этого
    Используется метод объекта PDO - lastInsertId
*/
$prodId = $connection->lastInsertId();
echo 'ID продукта: ', $prodId, '<br>';
```

- вставим данные другим способом:

```
<?php

// предположим, что нам уже доступна переменная $connection

// запрос для вставки данных (место для данных - ?)
$query = 'INSERT INTO products(name, description, price)
VALUES (?, ?, ?);';

// подготавливаем соединение к запросу
$insertStatement = $connection->prepare($query);

// исполняем запрос (передаем данные как простой массив)
$insertStatement->execute(['Smartphone Samsung', 'Another
nice smartphone!', 859.99]);
```

- множественная вставка данных:

```php
<?php

/*
    предположим, что нам уже доступна переменная $connection,
    а также переменная $prodId, куда записано значение
    первичного ключа некого продукта
*/

// запрос для вставки данных (место для данных - ?)
$query = 'INSERT INTO product_attributes (
    name, value, product_id
) VALUES (?, ?, ?), (?, ?, ?);';

// подготавливаем соединение к запросу
$insertStatement = $connection->prepare($query);

// исполняем запрос (передаем данные как простой массив)
$insertStatement->execute([
    'color', 'gray', $prodId, 'camera', '48 MP', $prodId
]);
```

- выборка данных:

```
<?php

// предположим, что нам уже доступна переменная $connection

// запрос для выборки данных
$query = 'SELECT id, name, description, price FROM
products;';

// подготавливаем соединение к запросу
$selectStatement = $connection->prepare($query);

// исполняем запрос
$selectStatement->execute();

// обрабатываем постепенно все полученные ряды таблицы
while ($row = $selectStatement->fetch()) {
    // выводим на печать данные ряда таблицы
    echo $row['name'], ' ', $row['price'], '<br>';
}
```

- выборка данных с условием:

```
<?php

// предположим, что нам уже доступна переменная $connection

// запрос для выборки данных (место для данных - ?)
$query = 'SELECT id, name, description, price FROM products
WHERE name = :name;';

// подготавливаем соединение к запросу
$selectStatement = $connection->prepare($query);

// исполняем запрос
$selectStatement->execute(['name' => 'Smartphone iPhone']);

// получим все ряды сразу - запишем их в массив
$products = $selectStatement->fetchAll();

// теперь можем работать с данными как с обычным массивом
foreach ($products as $product) {
    // выводим на печать данные ряда таблицы
    echo $product['name'], ' ', $product['price'], '<br>';
}
```

- обновление данных:

```php
<?php

// предположим, что нам уже доступна переменная $connection

// запрос для обновления данных
$query = 'UPDATE
    products
SET
    name = :update_value,
    price = price - 100
WHERE name = :name';

// подготавливаем соединение к запросу
$updateStatement = $connection->prepare($query);

// исполняем запрос
$updateStatement->execute([
    'update_value' => 'Smartphone iPhone (Discounted)',
    'name' => 'Smartphone iPhone'
]);
```

- удаление данных с условием:

```php
<?php

// предположим, что нам уже доступна переменная $connection

// запрос для удаления данных
$query = 'DELETE FROM products WHERE name = :name';

// подготавливаем соединение к запросу
$deleteStatement = $connection->prepare($query);

// исполняем запрос на удаление
$deleteStatement->execute([
    'name' => 'Smartphone Samsung'
]);
```

- удаление данных без условия (в таблице будут удалены вся ряды):

```php
<?php

// предположим, что нам уже доступна переменная $connection

// запрос для удаления данных
$query = 'DELETE FROM products;';

// подготавливаем соединение к запросу
$deleteStatement = $connection->prepare($query);

// исполняем запрос на удаление
$deleteStatement->execute();
```